Digital Systems Design with PLDs and FPGAs

Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science – Bangalore

Lecture-06
Controller Design

So welcome to this lecture on advanced digital system design in the course a digital system design with PLDs and FPGAs. The last lecture we have looked at how to go about designing a complex system like a CPU. We have taken an example of an 8 bit microprocessor. We have said that it cannot be defined in a flat way in one shot, it has to be partitioned into the functional blocks, which requires the my knowledge and you to decide on the interface between this block.

Then at the next level and at that level top level we have to decide the function timing all that. At the next level you will handle each block separately and try to design and go to the third level and then when we are kind of we end up with the known blocks like register, multiplexers, encounters, then we can stop you know that I need way of designing and it has been called design .

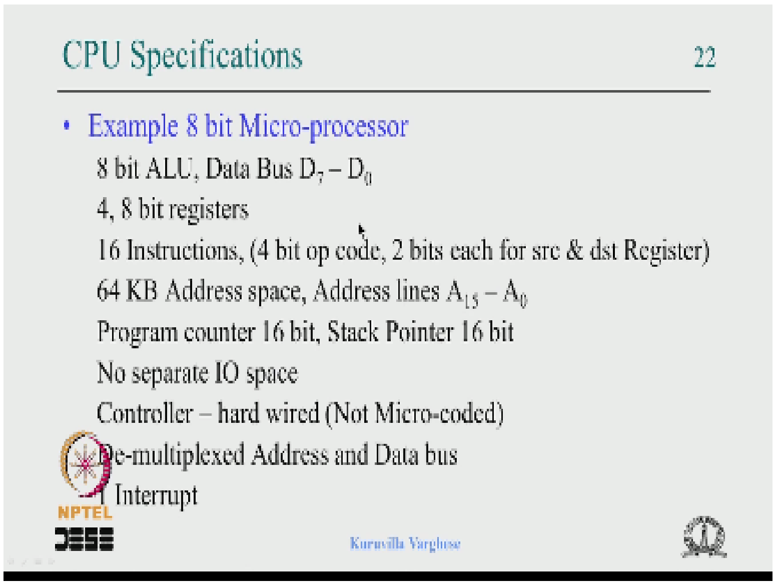
The same it is same what I am talking about . We have looked at the example of the CPU we have seen at the level 0, a block diagram level 1 partition and then for the level 2 we have picked up 2 blocks, one was the register CPU register, which is slightly simple. Then we have taken a program counter and designed it in detail and we have also discuss the division of the data path and the controller to bring clarity.

So that is what we have covered in the last lecture and today we are going to look at the behaviour of the controller and we will try to from the behaviour from that idea we will try to kind of come out with the structure for the controller and look at the properties and check what whether a generic architecture which fits for all cases, we can come out with such a thing.

You know from this we are working out little bit backwards from an example to the basic normally in a text book the other way is done and which sometime difficult to kind of to understand the beginning because we learn something formally and the structure and you try kind of later on apply or sometime the textbook does not show the serious application it only been shows with very simple cases.

And then it does not bring clarity that is why I am kind of going from net sample and kind of walking backwards and that bring clarity so that idea. So before turning todays apart on the controller we will have a brief look at the last lecture part, then we will move on to todays part.

(Refer Slide Time: 03:53)



CPU Specifications

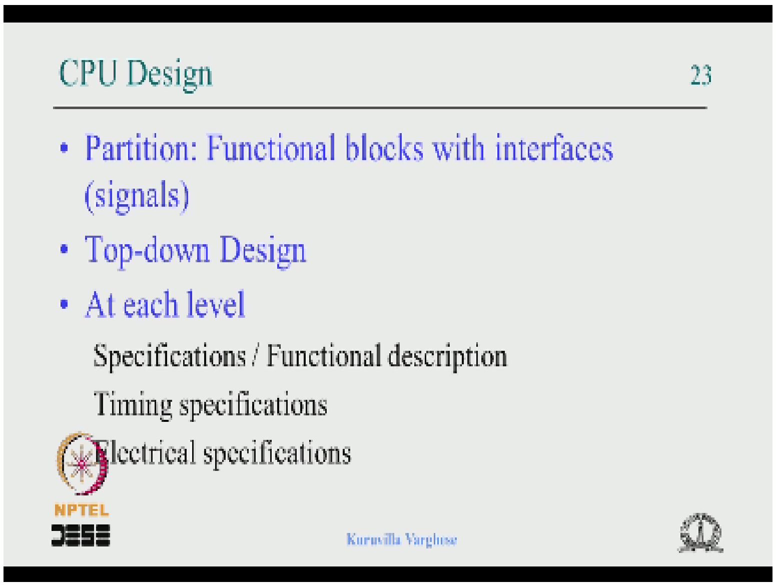
22

- Example 8 bit Micro-processor
  - 8 bit ALU, Data Bus $D_7 - D_0$
  - 4, 8 bit registers
  - 16 Instructions, (4 bit op code, 2 bits each for src & dst Register)
  - 64 KB Address space, Address lines $A_{15} - A_0$
  - Program counter 16 bit, Stack Pointer 16 bit
  - No separate IO space
  - Controller – hard wired (Not Micro-coded)
  - De-multiplexed Address and Data bus
  - 1 Interrupt

Kuruvilla Varghese

So let us move on to the slides of the last lecture and so we have looked at in a bit microprocessor but the thing was that it has 8 bit ALU data bus, 16 instructions, 64 address space, 64kb address space, which means 16 bit program counter and stack pointer the controller is hard wired not microcoded, no separate IO space and so on ok.

(Refer Slide Time: 04:24)



CPU Design

23

- Partition: Functional blocks with interfaces (signals)
- Top-down Design
- At each level

  Specifications / Functional description

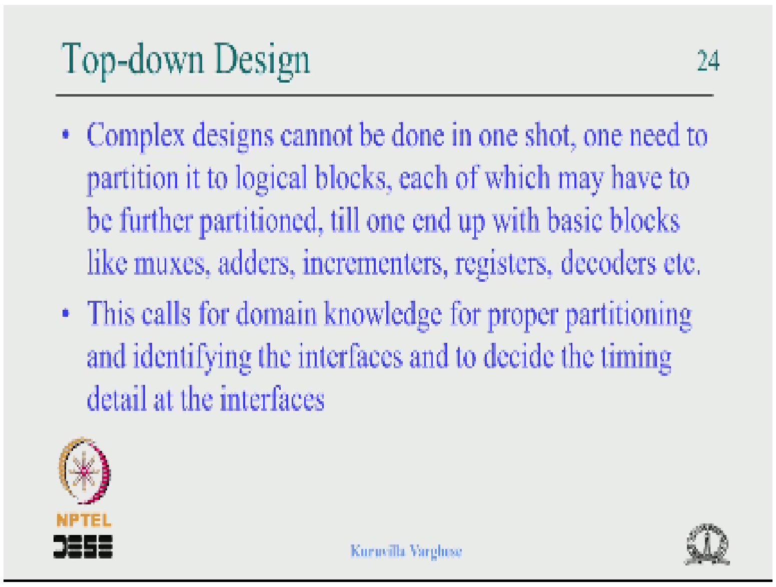
  Timing specifications

  Electrical specifications

Kuruvilla Varghese

And we said that we have to partition this in the blocks with interfaces we have to do a top down design at each level we have to handle the functional timing and electrical aspects of it.

(Refer Slide Time: 04:40)



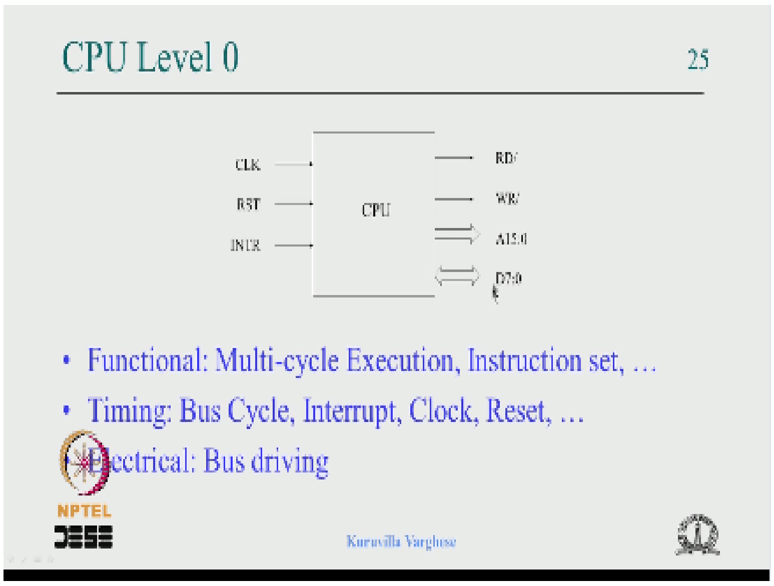
Top-down Design

24

- Complex designs cannot be done in one shot, one need to partition it to logical blocks, each of which may have to be further partitioned, till one end up with basic blocks like muxes, adders, incrementers, registers, decoders etc.
- This calls for domain knowledge for proper partitioning and identifying the interfaces and to decide the timing detail at the interfaces

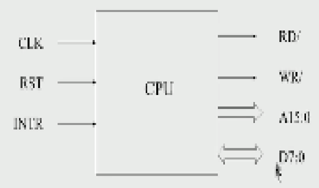
Kuruvilla Varghese

(Refer Slide Time: 04:43)



CPU Level 0

25



- Functional: Multi-cycle Execution, Instruction set, …
- Timing: Bus Cycle, Interrupt, Clock, Reset, …
- Electrical: Bus driving

Kuruvilla Varghese

And so we have looked at the top level it is a CPU with clock reset interrupt with input, read bar, write bar address is output and data as kind of bidirectional pin and we said this is a kind of we have decided not have to decide on the functionality we said this is there is no kind of pipelining, so it fetches then decode execute then it goes on fetches, so it makes it multiple clock cycle CPU for 1 instructions are executed take multiple clock cycle.

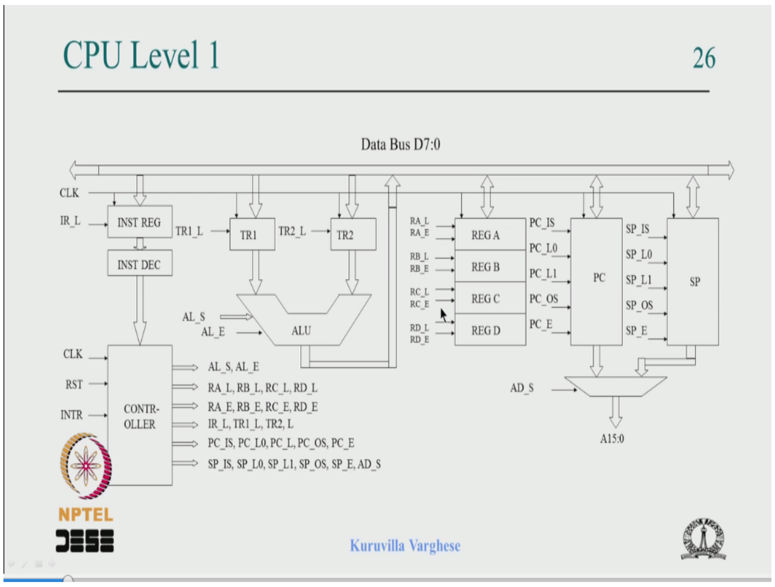
It is not very current design in the current nobody design like that, the CPU can be designed for a single clock cycle operation in most cases, but we are just fine too kind of basic, so we stay there and when it comes to the timings pack and of course we have to decide what are the instructions it is a supported what is instruction format and so on, but we will not get into that kind of details.

And our idea is in the process of you know design how the design process goes on and this you can refer to some book good reference books and books on microprocessor or computer Architecture, there is a good book by Hennessy and Patterson on computer architecture, two volumes are there, both deals with the CPU design in detail and here are the clock reset in that at some basic timing.

The clock may have some duty cycle the frequency the reset are from timing, interrupt will have you have to decide whether the level triggered let us get both and what is the timing password then so on and one important timing detail is a bus cycle how this bus is kind of behave whether the synchronous or asynchronous or how many clock cycles it take things and so on.

So and these are 2 kind of support, lot of load and this should have enough sourcing and sinking of the current.

(Refer Slide Time: 07:04)



CPU Level 1

26

Kuruvilla Varghese

And we have seen in the second level of partition level 1 where in there is an internal data bus, instruction, register, decoder, temporary register, ALU, a set of registers program counter and stack pointer. Everything is connected to a single bus in the current high performance CPU may have multiple buses. So that and all the high performance CPUs will have multiple buses.

That means the registers can feed the ALU, ALU can feedback the register and while that is going on instruction and come on to the instruction register and so on okay, or instruction free over ah Q, so many things can go in parallel and that we synchronise and so on, and we said there is a difference one is the data path where the data moves or the computation happens.

And there is a controller which sequence all the operation ok. So this partition is very good because when you functionally partition at you do not worry about the sequence of operation, you can concentrate on the functionality, concentrate on the interface and the timing details can be worked out later you know like what sequence this control signal come that can be decided later.

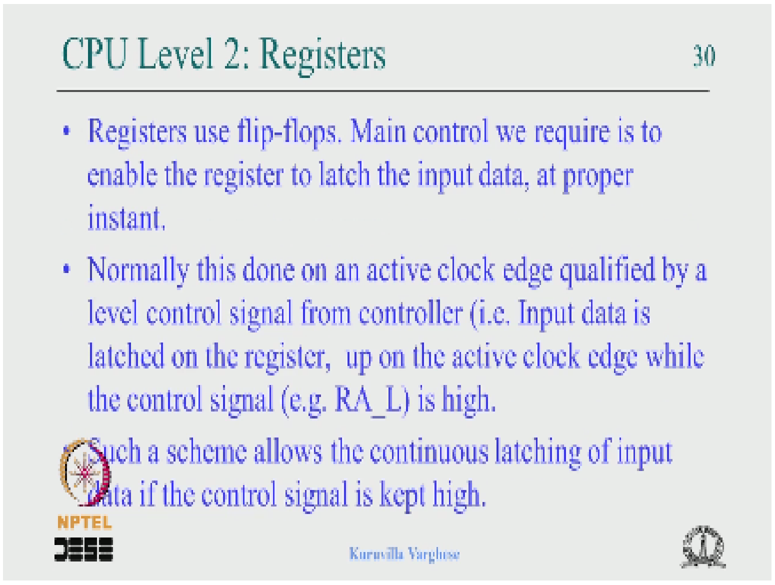
And the controllers job is to give the sequence operation, give the control signal, latch signal, enable signal and select the marks path to enable data into a particular path and so on. So that the job of the controller and we said in a synchronous case you need only one controller if everything is synchronous, but if there are kind of 2 concurrent activities which is not synchronous to each other.

Then a controller to handle 2 concurrent activities which are not synchronous ok and but in any case in a very complex scenario it is very difficult to manage with controller for modularity is of designing and debugging and testing and all that it will be ideal sometime, you are multiple controllers handling the multiple parts and a top-level controller coordinating with the bottom level controller.

So as in a top-down approach you can even imagine you could have the second level partition some of the blocks together having in a separate controller so on, that can be thought of one can think of a hierarchies controller does the top level controller which is in a coordinating the second level controllers and so on. So that is data path and controller and we have picked up an example of a register.

And we have designed it seen the design in details so that the other ideas register is that it should be connected or output should be connected to the database when it is enable the data should come similarly when the large signal from the controller comes and the clock comes the data latch.

(Refer Slide Time: 10:31)



CPU Level 2: Registers

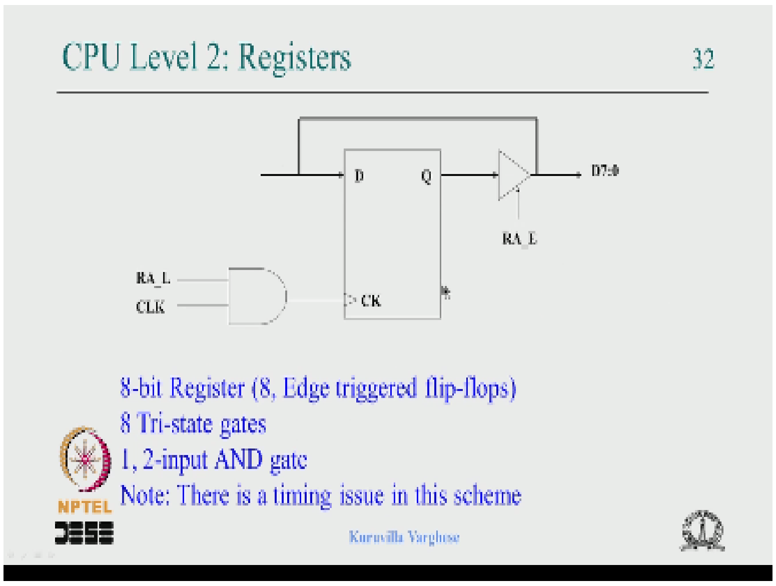
30

- Registers use flip-flops. Main control we require is to enable the register to latch the input data, at proper instant.
- Normally this done on an active clock edge qualified by a level control signal from controller (i.e. Input data is latched on the register, up on the active clock edge while the control signal (e.g. RA_L) is high.
- Such a scheme allows the continuous latching of input data if the control signal is kept high.

Kuruvilla Varghese

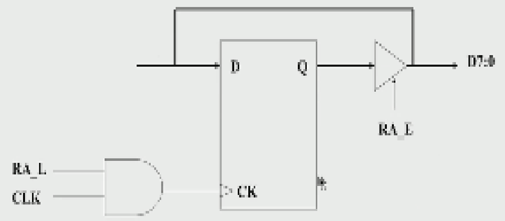
And can we have seen kind of circuit for that.

(Refer Slide Time: 10:34)



CPU Level 2: Registers

32



8-bit Register (8, Edge triggered flip-flops)
8 Tri-state gates
1, 2-input AND gate
Note: There is a timing issue in this scheme

Kuruvilla Varghese

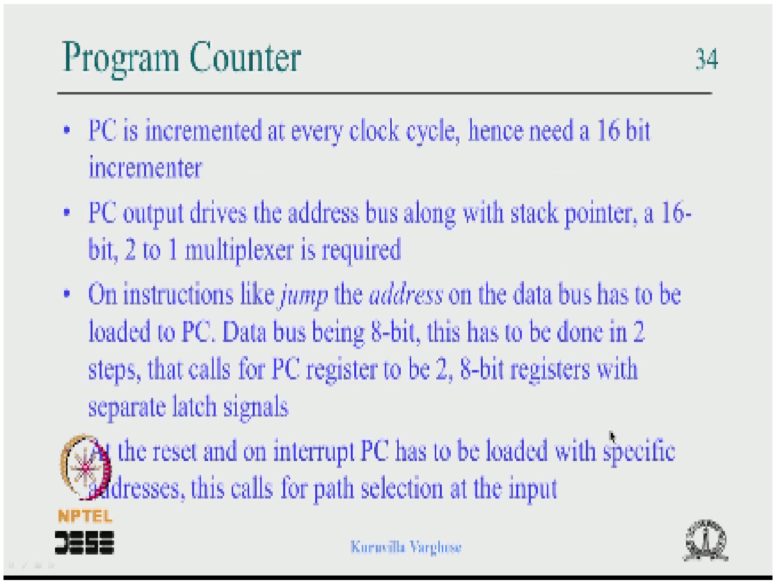
And say this is a possible circuit 8 flip flops clock is common and input and output is connected together to a Tri-state gate to the same data bus and the clock and the latch signal is under together, so that when the latch signal comes only then the clock comes. So at this level we know all the components flip flops, AND gate and registers design is over ok and that can be coded in VHDL or Verilog things like that.

But as a note says there is a little timing issue with it because we are sending it will not toper, there could be multiple pulses here and we will see that later, you know that if you can be handle later. So there is another possibility which avoids it then if you do not do anything with the crock path clock goes continuously and that problematic because every clock the data gets latch.

So we have to take care of that so the idea is to put a 2 to 1 mux and when the latch signal come the input goes directly when it is 0 it is re-circulated ok, that is basic idea. And the next thing we have looked at was a program counter which has various operation at the data when it jump or clock comes, the address has been latched in you know byte by byte.

When there is a call the return address is stored back again 16 bit register byte by byte it has store back and then the latch signal, so there are two 8 bit registers inside which are separated and this has to be handle properly, there are different input password from the data, one is an increment part, 1 is a reset address, one is interrupt address. So that is this input select, output select is when the both address need to be transfer to the data bus you have to select a byte by byte, so that is the address output select.

(Refer Slide Time: 12:46)



Program Counter

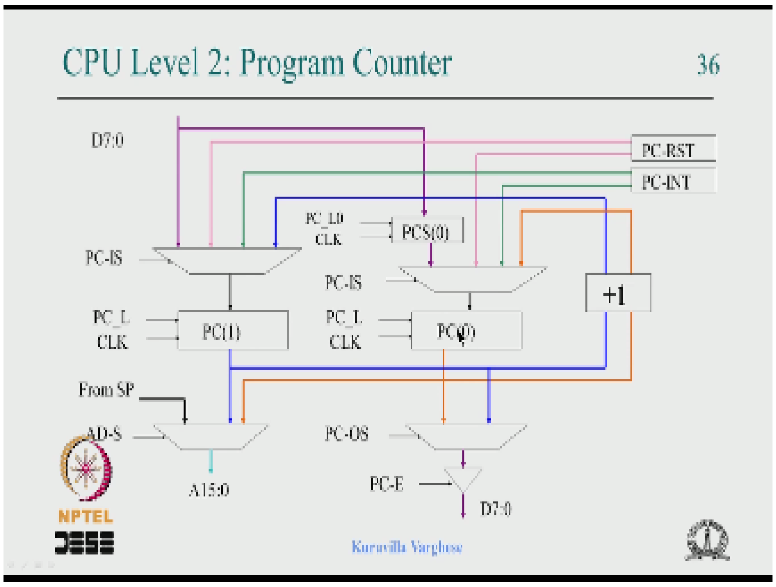
34

- PC is incremented at every clock cycle, hence need a 16 bit incrementer
- PC output drives the address bus along with stack pointer, a 16-bit, 2 to 1 multiplexer is required
- On instructions like *jump* the *address* on the data bus has to be loaded to PC. Data bus being 8-bit, this has to be done in 2 steps, that calls for PC register to be 2, 8-bit registers with separate latch signals
- At the reset and on interrupt PC has to be loaded with specific addresses, this calls for path selection at the input

Kuruvilla Varghese

Then enable for the output be connected to the data bus and that drives address first. And we have seen a in a possible design.

(Refer Slide Time: 12:51)



CPU Level 2: Program Counter

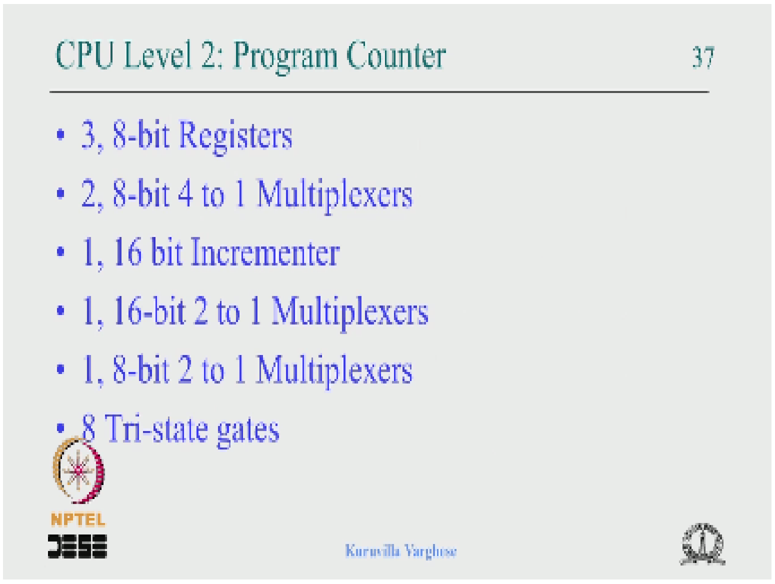
36

So you have two 8 bit registers which has latch same latch and we have a temporary register to transfer the least significant byte because address come least significant byte first and the most significant byte, so first it is transferred which is a latch signal separate from the controller, then when the MFP come this particular green path is selected and both together these significant byte.

And their significance by coming on the data bus is latch simultaneously on here, so the latch signal is common for both and when it is need to increment incremented this way and this red and blue path is selected and this latch pack. When the reset comes this address is loaded parallel in that comes that is loaded and when the program counter output need to go to the data bus.

This particular marks operates with the enable and this is the address bus being driven by the program counter. So it basically means three 8 bit register two 4-1 8 Max, 2-1 8 bit marks, 8 tri-state gate and one 16 bit 2 to 1 mux, the design is over. This can be easily coded ok. So that is how we design.

(Refer Slide Time: 14:19)



CPU Level 2: Program Counter

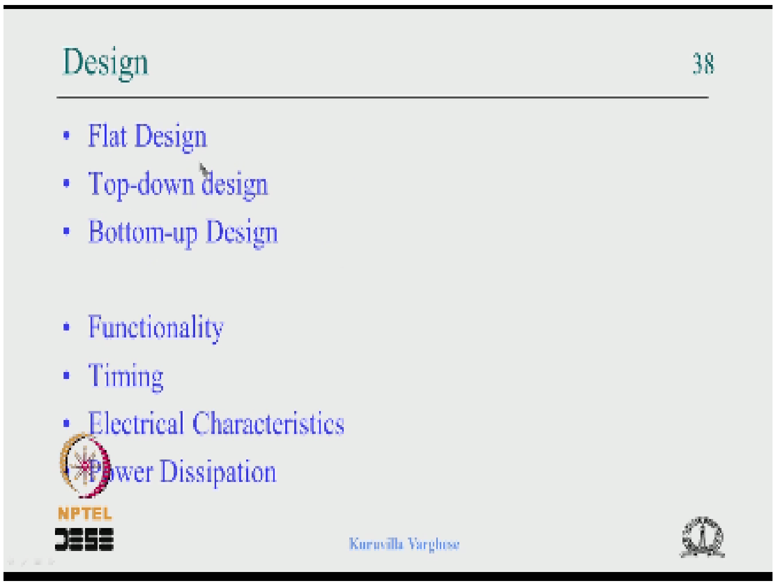
37

- 3, 8-bit Registers
- 2, 8-bit 4 to 1 Multiplexers
- 1, 16 bit Incrementer
- 1, 16-bit 2 to 1 Multiplexers
- 1, 8-bit 2 to 1 Multiplexers
- 8 Tri-state gates

NPTEL

Kuruvilla Varghese

(Refer Slide Time: 14:22)



Design

38

- Flat Design
- Top-down design
- Bottom-up Design

- Functionality
- Timing
- Electrical Characteristics
- Power Dissipation

NPTEL

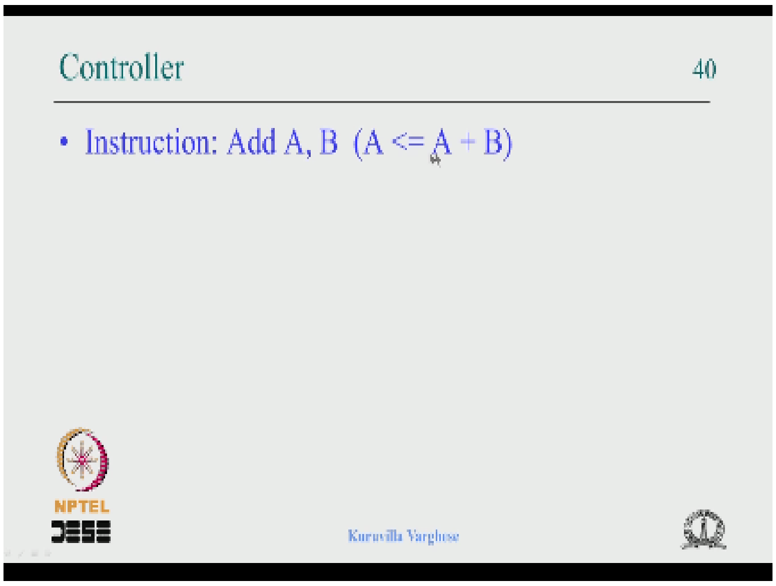
Kuruvilla Varghese

And we said so essentially the speed top down design in complexes and you should have the domain knowledge and you should at each level have to look at the functionality timing electrical characteristics maybe participation where it is critical. Nowadays everything most devices of mobile or which can be kind of transported and that needs a battery and which is better if power dispassion is control.

And there are many devices now sensors which is kind of installing one place which app which is supposed to work over long is the low power dissipation is very important. So let us come today's part. Today's part we are going to look at the controller this part of what is the behaviour of it, what is the structure of it may be possible introduction about how to go about designing.

If we know that is our focus today, so for the sake of illustration let us take an instruction because it basically a CPU is executing instruction and depending on the instruction these control signals are generated ok. So now I am going to take one instruction ok, so that can be extended to similar instruction and other instruction. So I am only kind of discussing a part of the game.

(Refer Slide Time: 16:06)



Controller

40

- Instruction: Add A, B (A <= A + B)

Kuruvilla Varghese

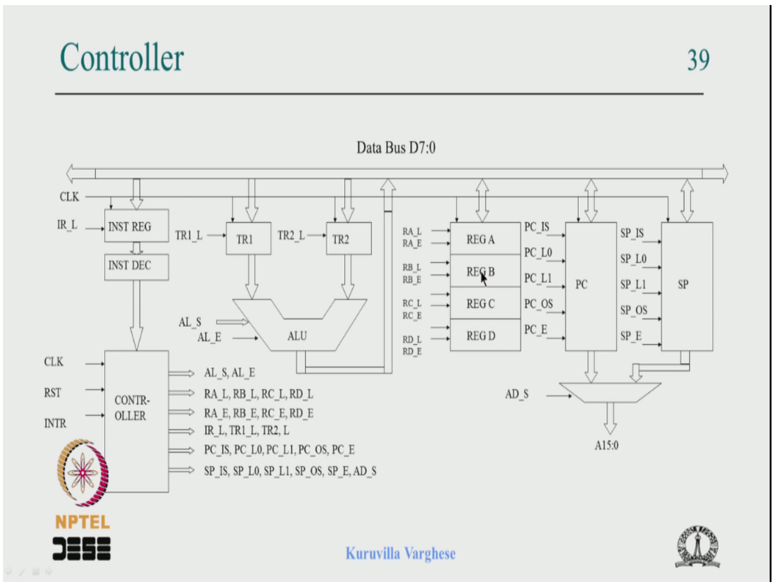
But that is enough to capture the idea. So let us take an instruction like add A and B ok, that means we are essentially adding at the content of the register A and the content of the register B and putting the result back at the content of A , so it is a kind of 2 Opran instruction like the intel, assembly language instruction or machine language instruction where when you say add A, B, the nothing but A+B **is** is loaded to A ok.

So let us see what happens when such an instruction is executed, so it is simple you move the content of A to the temporary register, then move the content of B to this temporary register 2, then you give select the ALU operation, that you say that it add operation ok. You can imagine there are multiple arithmetic and logic unit doing competition which is all parallel, which is more marks.

And you can think of this is selecting the path in the marks ok. This is a select line of the marks which is choosing the connecting the output of the appropriate block to the output of the ALU ok and this enable is a tri-state gate because only one device can drive the bus. So that as we enable with the result is so you give the operations selection, then you enable it, then register A is loaded with that result ok.

So that is a kind of macro step of adding A and B, but let us look at little more detail because our idea is the controller has to generate appropriate control signal. So we have clarity in terms of control signal ok. So the first operation is moving the content of A to the temporary register one ok. So what we need to do that we have to enable the register A output. So give a one to register AE which is register enable, then the data comes on the data bus ok.

(Refer Slide Time: 16:46)



And that goes to all inputs, all the register input goes, but then the controller has to give a pulse to the TR1L which is the temporary register 1 latch signal and it has to be in synchronous with the clock very important, then the clock edge comes and the data get latched, ok. Unless it is not synchronous it is not going to happen. So when it is one when it high the active clock edge should come ok, that should be taken care.

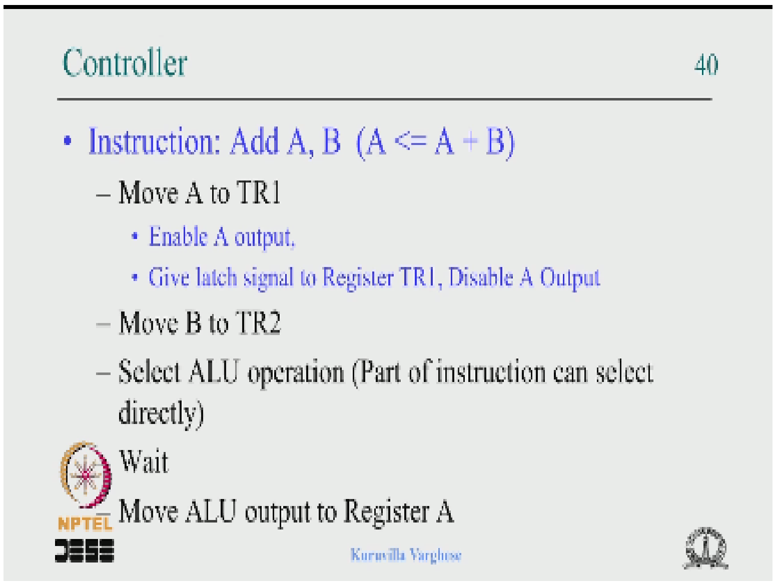
And then you disable okay, so that is a game. Now I can tell you a very simple idea. So what we should do that we give pulse to RAE and we can give the same pulse to the TR1L, assume that is asynchronous with the clock. Then the A value is enable because the pulse and it comes here and the clock edge comes to get latch and the pulse goes to 0, this disable, ok. Similarly you move the register B content into the temporary register to enable A give the same bus to the latch signal clock comes, it gets latched.

And when the bus goes 0, this is cut off ok. Now specify the add operation and that can be some bitch from the instruction directly there is may not be kind of decoding is not required, you can kind of maybe you can kind of you know encode, if you need in another fashion but may not be required same pattern can be given here from the opcode and that can select the operation or there is a kind of a further encoder which can goes into some less number of bits which we can come here and select this operation.

And this is enable after top period of weight and this is enabled and that same signal is given to the register, a latch same signal means the control will generate, at the same time another signal of the same timing and so it is enabled and on the active clock edge that is large and when the clock goes 0 that is disabled and add operation is done. Also A to tier 1, B to tier 2 add operation enable and result back to A. So enable is given a pass, latch is given a pass up on the clock is moved, it is disable.

It is a BE given a pass, Tr2L is given a pass and up on the clock it gets latched here and that is disable and this is given a selection, this is enable dame passes, similar passes coming at the latch signal and the data comes here it gets latched and it gets disable automatically.

(Refer Slide Time: 21:33)



Controller

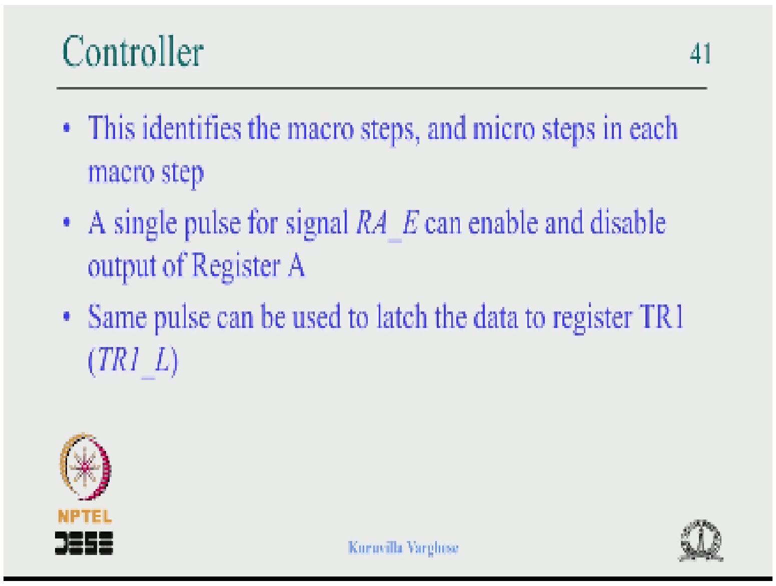
40

- Instruction: Add A, B (A <= A + B)
  - Move A to TR1
    - Enable A output,
    - Give latch signal to Register TR1, Disable A Output
  - Move B to TR2
  - Select ALU operation (Part of instruction can select directly)
  - Wait
  - Move ALU output to Register A

Kuruvilla Varghese

So that is a kind of sequence so I have written that here in detail move A to Tier 1, move B to tier 2, ALU operation wait and ALU operation output to register A ok.

(Refer Slide Time: 21:44)



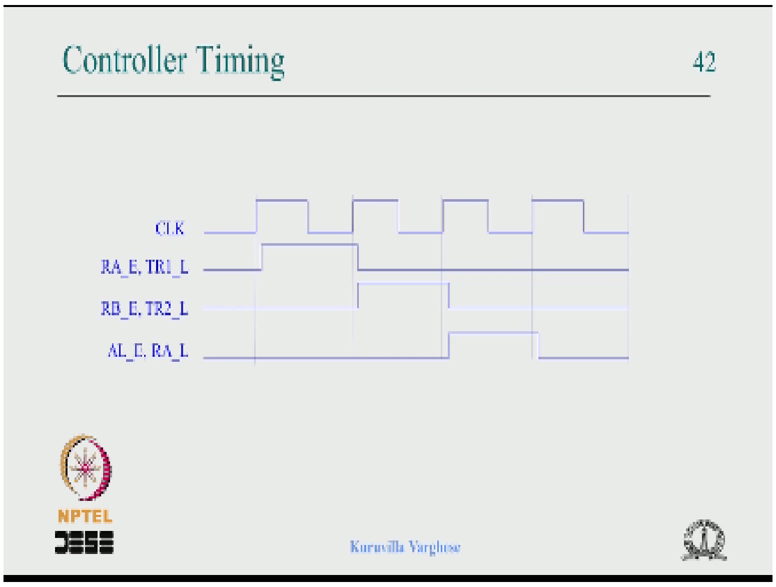
Controller

41

- This identifies the macro steps, and micro steps in each macro step
- A single pulse for signal *RA_E* can enable and disable output of Register A
- Same pulse can be used to latch the data to register TR1 (*TR1_L*)

Kuruvilla Varghese

(Refer Slide Time: 21:06)



Controller Timing

So now what I am showing is I am showing a possible timing waveforms which will achieve this will assume there is a clock you know going like that which is a as in a square wave clock. Now this is the timing ok. These are two separate output instead of you know drawing it separately I am showing it together. This is separate output coming from the controller.

This is not the same single, do not assume that it is same signal. It is 2 separate signal with the same pulse ok. So A enable and tier 1 L get similar pulses like it comes with the delay with respect to the active clock head like that. So at this point the data is enable, data is going on the data bus. So it has enough time before the active clock edge has come. So if you go back to this like when you enable something it take some time for the Tri-state get enable.

And propagated and set up at the input of the register then only the clock end latches. So that is taken care of by this kind of timing. So we are enabling it here at this point data start flowing and will be ready at the input and when the clock comes later date gets latched and the output is 0 ok. Similarly the B is enabled at this point and in the temporary just do it is lathed and B is disable.

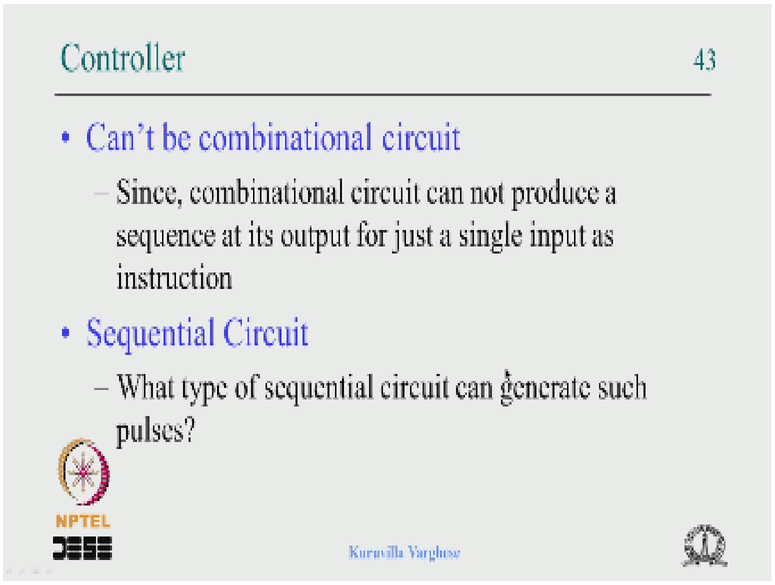
And I am not specifying the ALE operation but that can be taken from the opcode with a **a** encoding if needed and this is ALEs enable and ALE is enable and that data is coming here and get latched today and it is disable ok. This simple scheme achieve this operation instruction add A, B ok. Now the question is of course you know there are other instructions that one has to maybe if say you can imagine add BC, add AB everything is same.

So all that can be easily captured by this behaviour, maybe subtract, again things are same only thing is that the subtract operation is difference, so the controller which was designed the part of the control of which does can be generalized to do many other things ok. So that is the basic idea, for you in might look little convoluted that I am kind of orchestrating scheme very clearly.

I am pushing the edge little bit 2 to the right hand side, so that you know the clock heads come here, but you will you will see that what we practically come out with will match this. At this point you take and you may think it is a kind of manipulation but it will turn out to be true correctly like the structure will manifest this are ok and also you should know that we are kind of going to have everything in terms of integral number of the clock cycle ok.

So the basic unit of control is a clock period ok and nothing less than that. So we will have a pulse of at least one top period not have proper, so everything will be decided per clock cycle that should be kept in mind, now the questions is that can the controller be a combination circuits ok, because we have studied combinational and sequential circuits.

(Refer Slide Time: 25:48)



Controller 43

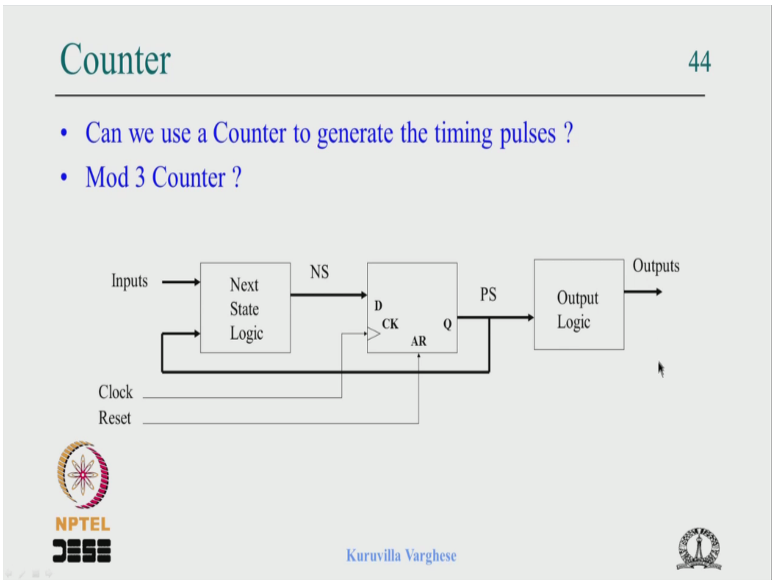
- Can't be combinational circuit
  - Since, combinational circuit can not produce a sequence at its output for just a single input as instruction
- Sequential Circuit
  - What type of sequential circuit can generate such pulses?

Kuruvilla Varghese

So we know that combination care if given a input say N input it can produce the M output, but it cannot definitely produce given N input to a combinational circuit, it cannot produce the pulse like this you know every cock like for 3 o'clock period. So this a controller cannot be a combinational circuit, it has to be sequential circuits ok. It is like given an input it generate A sequence of pulses okay.

In this case it looks very neat, one pass then the second pass and third pass and so on. So the question is that what type of sequential circuit can regenerate such a pulse okay like the answer is that unfortunately we have studied only the counter now synchronous counter, so I want to straightly ask can we have a synchronous counter generating this kind of pulses, you know that is a question we are asking.

(Refer Slide Time: 26:51)



Counter

44

- Can we use a Counter to generate the timing pulses ?
- Mod 3 Counter ?

NPTEL

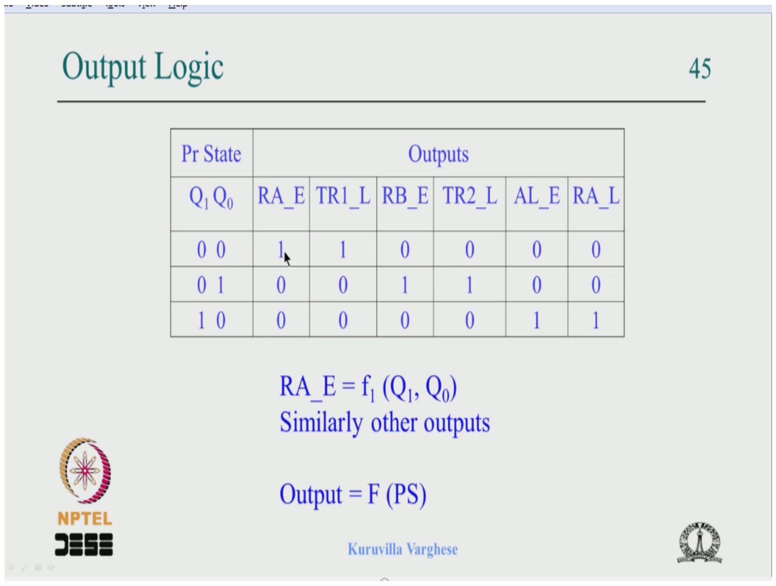
Kuruvilla Varghese

Can we use a counter to generate timing pulses. We have Mod 3 counter, like we have the counter which is designed like this you know you have a state flip flops which is decoded the input to generate the next count, can we have a counter generating a Mod 3 counter generating this kind of pulses ok. Like thing the Mod 3 counter will count 0, 1, and 2 and it has 2 bits ok.

So what we are require is separate output, ok. Now so why not we decode this particular pulse from that 0, 0 count ok and VD code this 2 pulses from 0,1 and VD code this 1,0 in these output from other count 1, 0, so that is basic idea. So we take a counter and what we do is that we do not need 0,0,0,1, 1,0 we need some output. So what we do is that we protect combinational circuits here.

And we generate output as a d code of the states ok. So when it is 0,0 will generate 2 output 1,0,1 and other 2 outputs 1,0 another 2 outputs ok. Now we have to design this output logic this is call output logic which the input as a present state and output is the various output we need. So we can write the truth table for the output logic in this case. So we have input in present state.

(Refer Slide Time: 28:40)



Output Logic

45

Pr State	Outputs					
$Q_1 Q_0$	RA_E	TR1_L	RB_E	TR2_L	AL_E	RA_L
0 0	1	1	0	0	0	0
0 1	0	0	1	1	0	0
1 0	0	0	0	0	1	1

$RA\_E = f_1 (Q_1, Q_0)$
Similarly other outputs

Output = F (PS)

Kuruvilla Varghese

In this case 2 bit Q1, Q0 and there are 6 output ok. I said for convenience I have shown these has together but it has to be separate because luckily in this case this has similar pulses, but if you are adding b2c then TR1 latch and B will have the same. So it has to be separate, so we know that in 0,0 at least for this case these 2 have 1,0,1, this register B enabled and the TR2 as 1 and in the 1, 0 ALE enable.

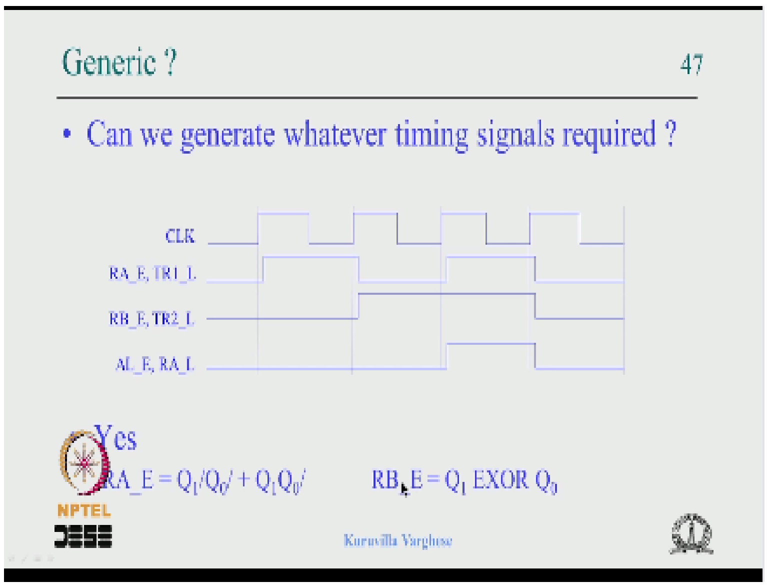
And the register A latched 1 ok. Now it is very simple for each output will pick up one form at the mid terms in terms of Q1, Q0 optimise it come out with the equation. So like for an example we take this register A enable that you will get us a function of Q1 and Q0 with is a present state. Similarly you can work out for all other 5 outputs. So we can say output is a function of the present state.

Now so the output is decoded from the present state and then we say output is a function of the present state ok. So that is all we generate this time because it is complex because we are only considering now one instruction, we have to kind of combine all similar instruction and there maybe this Mod3 business this may not work, it may my not, it may have to count in a very complex way making branches and so on depending on the various input.

So you can imagine these are the various instruction, various group of instruction and depending on that will make a lot of transition and each transition is decoded each state is decoded to produce output by this output logic. So from a counter we have graduated to the next level having another logic and output which is decoding the present state and producing the various outputs ok.

Now you see I have the before coming to this part this kind of delay know that this enable and the latches delayed by sometime with respect to active clock head looked at artificial, but now you look at the diagram we are putting a logic here, so that clock comes and this present state will change after PCQtime delay. Now the output will come after the clock edge after PCQ and output logic ok.

(Refer Slide Time: 31:50)



Generic ?

47

- Can we generate whatever timing signals required ?

Yes

RA_E = $Q_1'Q_0' + Q_1Q_0'$ RB_E = Q_1 EXOR Q_0

Kuruvilla Varghese

So when you implement when you use such a structure to generate this kind of output then you can see that there is a delay with respect to the active clock edge which is nothing but PCQ+T output logic, so everything will be delayed and which is required unless it is delayed this edge cannot you know latch the data, so it is very important that there is such a delay and that is automatically by this particular structure ok.

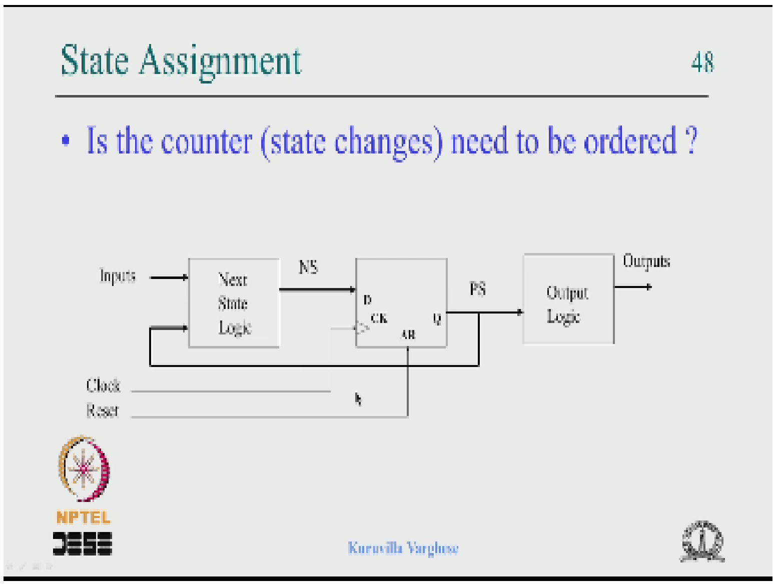
Now the next question is that we have come out with a structure which is an extension of asynchronous counter we have added an output logic to the synchronous counter. The next question we ask is fine it can do add A+B can it do everything what we require, can we generate whatever timing signals required, say exam let us forget about a particular instruction, let us post a question that ok.

Now the register A enable and Tr1L as a different pulse signal or timing. So at the first clock period it is high, then it is 0, then in the third clock period it is high. Similarly for the second set it is high for consecutive 2 clock cycle with the second clock cycle and third clock cycle. Let us forget about what instruction implement I am only passing a hypothetical question .

You can you confirm any kind of pulses okay, the questions that can we generate whatever timing signals required. So if you look at this particular structure we can say very definitely has ok because we are decoding the person state. So it is not necessary that you have to decode like a pulse one in the first clocks in o clock cycle 0, 0 like you can say let us make this decode this particular signal as having 1 in the 0 0 and 1 in 0,1,0.

So then that is done. So it is very simple that yes it is possible A register A a enabled is nothing but Q1 bar Q0, the 0, 0 are say Q1, Q0 bar that is 1, 0 ok. This is 0001 10, similarly when you come here it is nothing but 0 1 and 1 0. So you get equation for that which is enabled in that case Q1 bar Q0 bar Q1, Q0 bar which will be kind of you know is simplified Q0 bar ok.

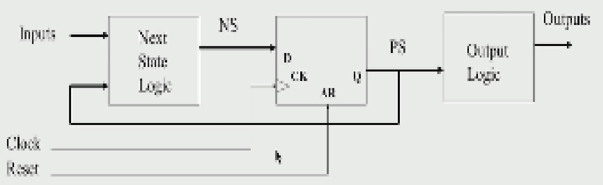
(Refer Slide Time: 34: 56)



State Assignment

48

- Is the counter (state changes) need to be ordered ?



NPTEL

Kuruvilla Varghese

Similarly register B enabled if you look at this it is Q1, EXOR Q0 bar. So what I want to convey is that this particular structure is very generic know whatever maybe the timing requirement in terms of clock period. This structure can generate it you know. You do not worry saying the inspection is different or it is a very complex think it is something with the program counter the call has come.

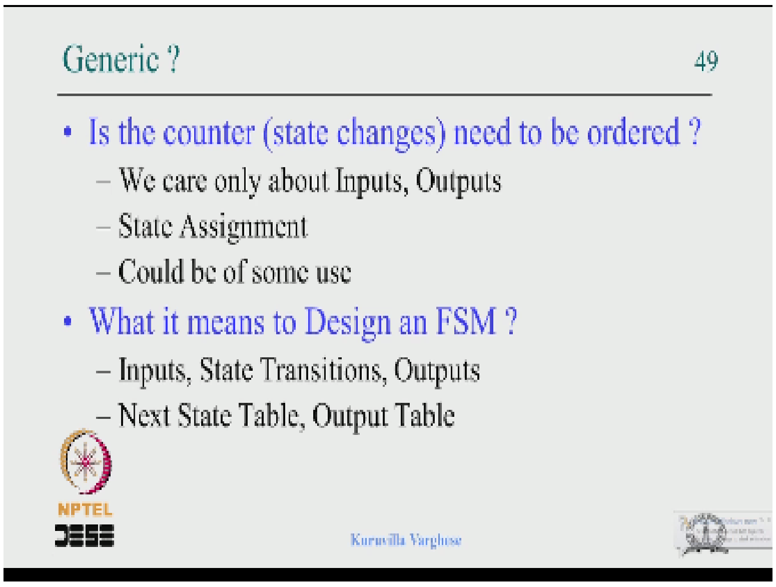
So the current affairs has been pushed to the start, new address need to be stored, how this can be all this possible and all that ok. So maybe the diagram I have shown are required some modification for with modification probably I do not know maybe some shadow register are required, but whatever it is, it can be done it from my controller perspective number of it requires generate different type of causes.

And this structure is able to generate it. So that what is comes to is that when you talk about a controller this structure of this architecture is very generic to be used for any controller like you have a multiplayer you need a controller and then this can be used only thing is that depending on the application. This next state logic will change because depending on the input an application this transition will change.

And at each state the output number of output and the timing of opportunities through this logic will change. So depending on the problems depending on the timing this next state logical change, output logical change, number of flip flops will change, the transaction will change, but we can say this is the architecture of a controller general, architecture of a controller.

Only thing we have to decide for a particular problem is this how many bits are required what is the next state logic, what is the output logic ok. So that is the conclusion, so this is called a finite state machine because this machine goes through the finite number of state, it transit through the find number of states a complex when it is not that it is like the counter it can take branding decisions comes to a state.

(Refer Slide Time: 37:33)



Generic ?

49

- Is the counter (state changes) need to be ordered ?
  - We care only about Inputs, Outputs
  - State Assignment
  - Could be of some use
- What it means to Design an FSM ?
  - Inputs, State Transitions, Outputs
  - Next State Table, Output Table

NPTEL

Kuruvilla Varghese

Then depending on input we can go branch to a one particular state or branch to another state, another set of state and so on ok. So the transition can be complex unlike in counter so that they said it is it is very generic. The next question is that is a counter need to be ordered the uses state we need to order that means we said for adding section we set count 0 0 0 1 and 1 0, does it have to be ordered. Can we call 3, 0 and 1 or 3, 0, 1, 2.

Yes it does not matter it can transit at the state does not matter because we are worried about the output and even some set of input we need a sequence of output and we do not care too much about how this counter state part keep track of the state as long as it is this thing state for each step then we can decode accordingly. So it is not very important how the state particular numerical value of the state ok need not be very order, it can be orbitary.

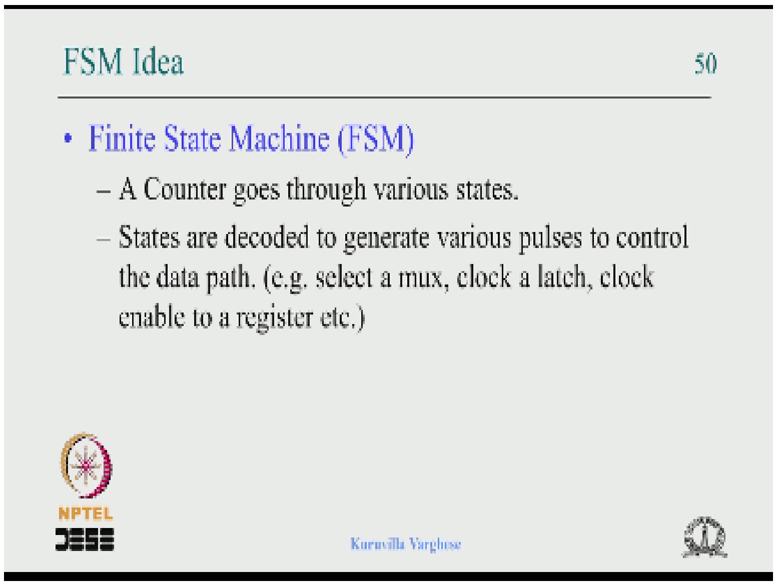
So that problem is cal state assignment ok. So we care only about input and output it can be assigned with a useful values and that could be of some use ok, maybe question is that if you assign the state in a particular order maybe this logic or minimise or this logic at this particular logic at minimise ok. So the state transition and the numerical values of no consequence.

We can be coded as we can change the decoding of truth table of next logic and truth table of output logic to reflect the state you are a signing the designer of assigning to reach some benefit out of that assignment like having the minimal area for the logic and so on ok or to solve some timing problem ok. And so what is what it means to design and as I said this is called a finite state machine or FSM ok.

So that is the expansion of this finite state machine. So basically it mean that you have some input depending on the input the transition happens state transitions happens have because it had a beam section, we have 3 states which is transistor and each state generate some output. So this is a specification ok. This is algorithm and we while designing at the end of a design this next state logic and output logic.

So what it means to design FSM is that, your specification of the input the state transition and output are the sequence and then from there you come and design the next state table and output table ok. So that is a basic what it means to design on a finite state machine.

(Refer Slide Time: 40:46)


FSM Idea

50

- Finite State Machine (FSM)
  - A Counter goes through various states.
  - States are decoded to generate various pulses to control the data path. (e.g. select a mux, clock a latch, clock enable to a register etc.)

NPTEL

Kuruvilla Varghese

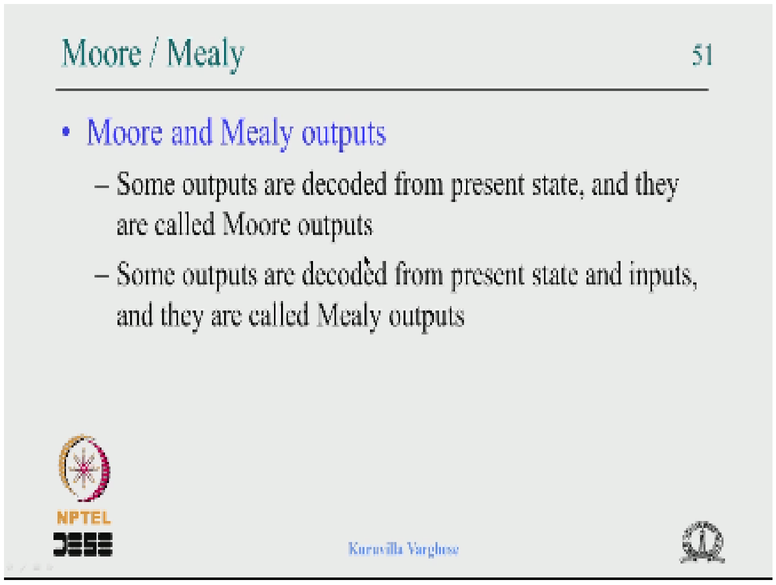
So the basic finite state machine are controller and roller is nothing but a finite state machine, idea is that there is a counterpart with go through various it is a material whether it goes through in a particular sequence and states a decoder to generate various pulses to control the data path ok, maybe this control signal will select the marks, give a clock signal to a latch, will enable, talk to adjust increment counter and so on ok.

That is what are these output signals are used for. They do the sequencing operation of the datapath like as I said enabling something in the register incrementing counter, selecting a multiplexer path and so on, because these are the important thing we do with the sequencing pulses ok. So let us come back to the slide, so here we look at this particular architecture and you see here the output is generated as a decode of the present state ok.

Now in some cases it is convenient we will see that later what is advantage of doing that or when we can do that sometime it is convenient to generate the output not only has a decode of present state, but as a function of both present state and input, so you can imagine this input is coming here and you are decode the output as a function of present state and input ok.

So it may happen that some output the decoded from the present state, from outputs are decoded from the present state and the inputs ok. Now when you decode an output from the present state is called Moore machine output this is after the mathematician Moore and when you decode an output as a function of present state and input then is called Mealy machine ok.

(Refer Slide Time: 43:10)



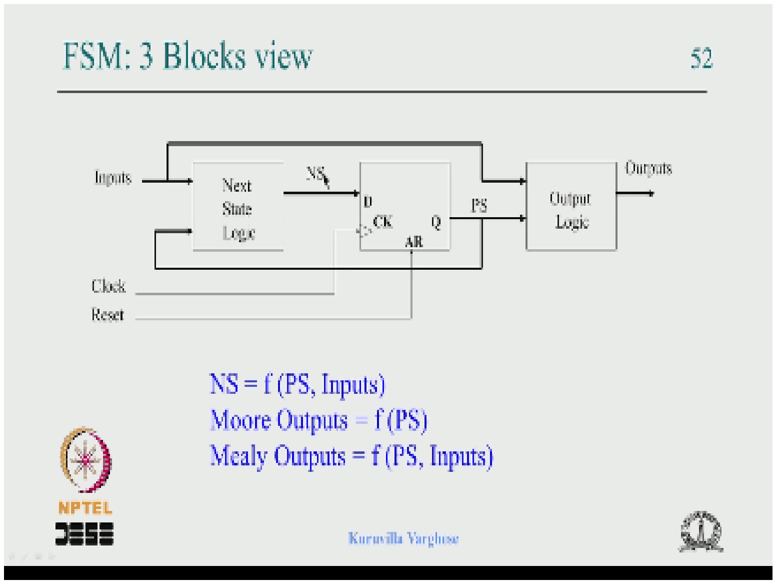
Moore / Mealy 51

- Moore and Mealy outputs
  - Some outputs are decoded from present state, and they are called Moore outputs
  - Some outputs are decoded from present state and inputs, and they are called Mealy outputs

Kuruvilla Varghese

So we have two types of output, one is a Mealy output when output is decoded from the present state and input and Moore output when the output is decoded from the present state only. So we can modify that earlier architecture or structure that where the output logic gets not only is the present state but also the input ok so.

(Refer Slide Time: 43:40)



FSM: 3 Blocks view 52

NS = f (PS, Inputs)
Moore Outputs = f (PS)
Mealy Outputs = f (PS, Inputs)

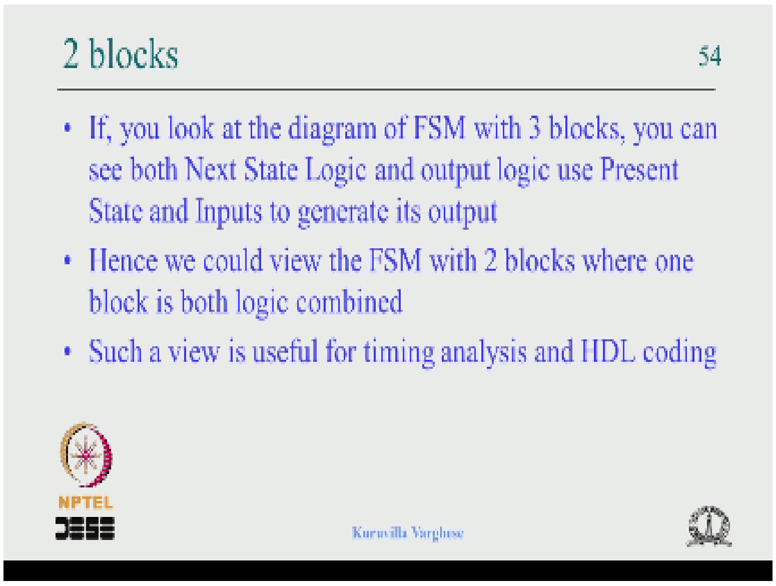
Kuruvilla Varghese

So we say here the when you say the decoding the next state logic decode the next state as a function of present state and input and output logic when generate some output as a function of present state which are called Moore output and the output logic decode some output as a function of present state and inputs which is called Mealy output ok. Now I should mention that most of the text books you have studied in the undergraduate program talks about Mealy machine and Moore machine ok.

It is kind of very kind of simplicity you many times you would have noticed that the finite state machine you have kind of seen example is just with many times one input and one output okay. In such a case it is very easy to call the Mealy machine because there is only one output which is Mealy output or 1 output which is Moore output. So you can call a Moore machine and Mealy machine.

But in practical cases the controller or state machine has lot of input lot of output , so you can talk kind of you know Mealy machine and Moore machine because if even if there is one output which is Mealy then you will end up calling that the machine that could confuse design. So let us keep this kind of distinction. Now tell one thing the notice is that now you look at this next state logic.

The input the next year logic is the present state and input and it decode the next state ok. Now if you look at the output logic the input it is a process state and the input and it decodes out ok. Now why do not we like if you look at the inputs of same, so why do not we combine this together in a block and we call logic the output of the logic is next state and outputs ok.
(Refer Slide Time: 45:56)



2 blocks

54

- If, you look at the diagram of FSM with 3 blocks, you can see both Next State Logic and output logic use Present State and Inputs to generate its output
- Hence we could view the FSM with 2 blocks where one block is both logic combined
- Such a view is useful for timing analysis and HDL coding

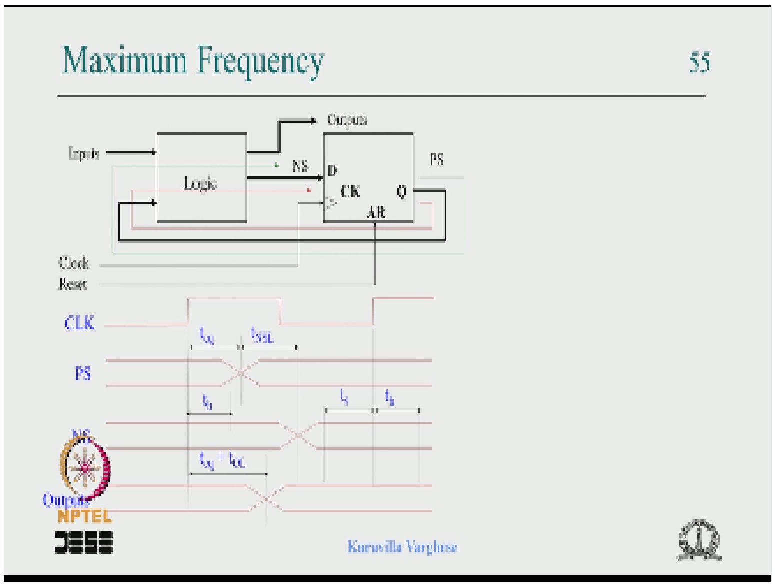
NPTEL

Kuruvilla Varghese

So we get a 2 block view which is nothing but we are combining the next state logic and output logic the logic looks at the input and the present state, it decodes the next state and is also decode output, such a very useful view because for when we analyse the timing I when we do the VHDL coding this kind of 3 blocks you on the to block you is useful because when we analyse or maybe you are you doing the timing analysis this is easier to look because it is much simpler , it captures all the kind of data various parts together.

And when VHDL code maybe you will call this separate, this separate and this separate but that would be advantage in coding the logic together you know various ways. So these 2 views are very useful, please keep this in mind and this to use and as I said any finite state machine the architecture is same only the logic differs or depending on the application, depending on the input output timing the logic changes and the number of flip flops and transition change , the architecture essentially remain same ok.

(Refer Slide Time: 47:12)

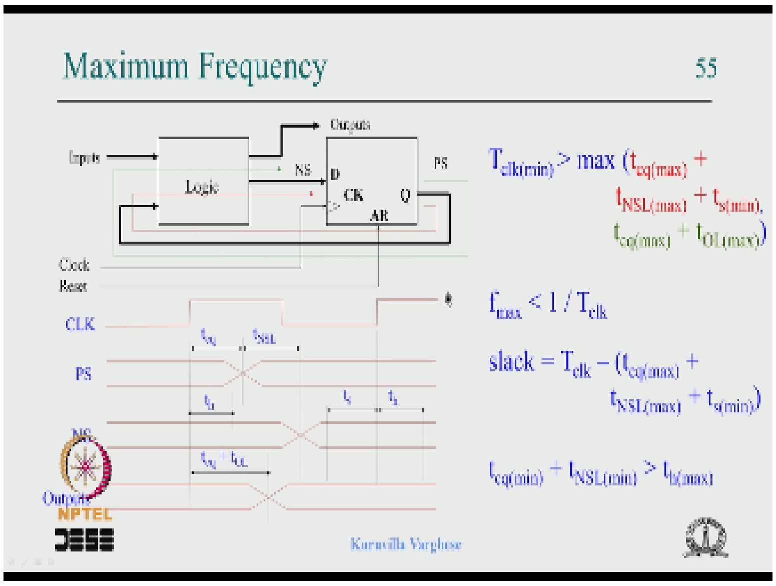


So that is the description of the 2 block . Now let us look at the timing issue of the state machine the finite state machine we will use the 2 block view because it is much more kind of concise. So one important thing to look is that there are now 2 parts from a register output through the next state logic to the register, other register or the same thing we cannot say,

I give there are 3 flip flops 3 inputs, 3 output then it can be 9, 3x3 9 this in path maximum ok. But very important things notices that there is another part from through the flip flop through the output logic to the output ok. So when we consider the maximum clock frequency if you consider only the clock to clock it is not useful because the clock comes then it propagates here, propagate here and the set up time is match TCQ+tNSL logic Ts ah should be kind of less than the clock period.

The clock period should be greater than this, but suppose if the Tcq_Toutput logic is even greater than that the transition will happen correctly but the output would nto appear the output.

(Refer Slide Time: 48:46)



Maximum Frequency

55

$T_{clk(min)} > \max(t_{cq(max)} + t_{NSL(max)} + t_{s(max)}, t_{cq(max)} + t_{OL(max)})$

$f_{max} < 1 / T_{clk}$

$slack = T_{clk} - (t_{cq(max)} + t_{NSL(max)} + t_{s(min)})$

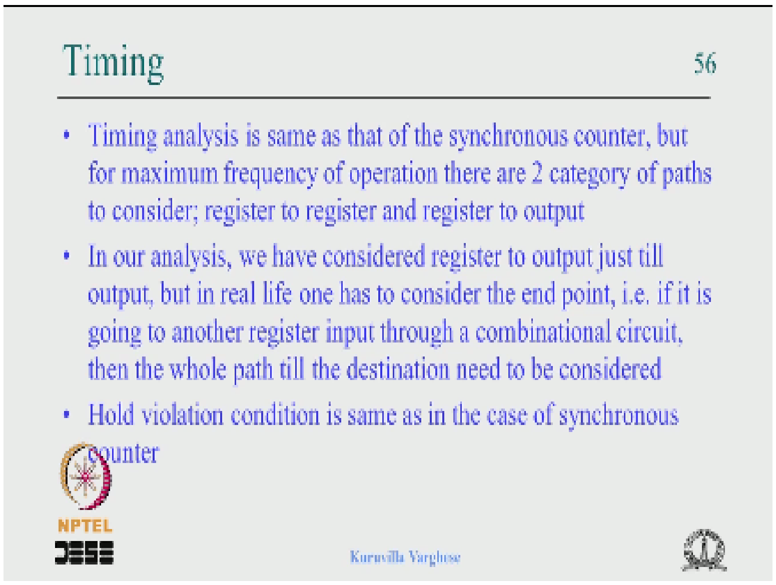
$t_{cq(min)} + t_{NSL(min)} > t_{h(max)}$

So what I say that we will say the we will say the clock period the minimum clock period should be maximum of the Tcq+tNSK logic+Tsetup time, or Tcq+Tout logic. This output has to appear down ok, and normally in most cases it is kind of close together we can imagine that this is kind of a larger than this quantity, but one issues is there we do not know where this output is going.

Supposed if this output is going through a combinational logic and going to another register as part of the control then in that case we have to analyse the path from this particular register through this output logic through the combinational logic to the destination register ok. So in the since I do not have any of affair information about that has returned that but in real life that could be the like Tlogic+T setup time time of the destination register may be coming here that you should not forget ok.

So the slacking the margin we give and the whole time violation is that the Tcq min Tcq min+T next state logic min should be greater than the whole value ok. So that they like when the clock comes because of the clock the data this point changes but that should be only after the whole time as I said it is it is with respective same clock head ok. So that is the essence of it.

(Refer Slide Time: 50:37)



Timing

56

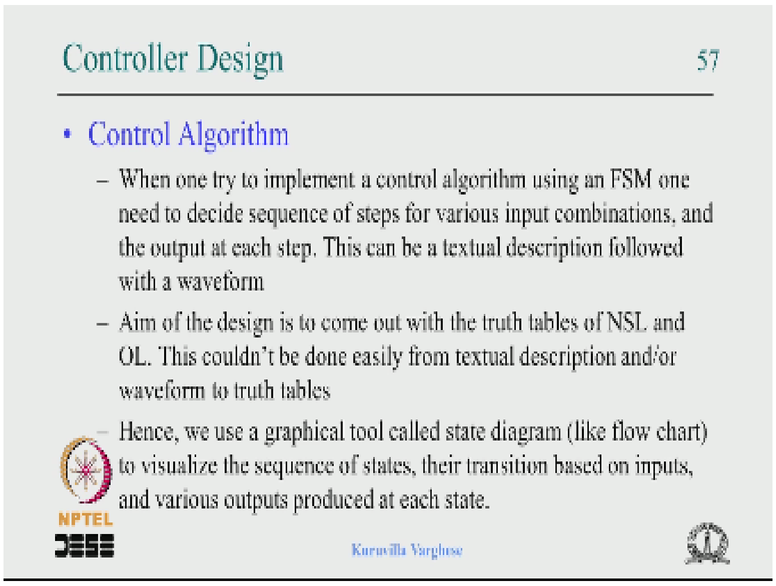
- Timing analysis is same as that of the synchronous counter, but for maximum frequency of operation there are 2 category of paths to consider; register to register and register to output
- In our analysis, we have considered register to output just till output, but in real life one has to consider the end point, i.e. if it is going to another register input through a combinational circuit, then the whole path till the destination need to be considered
- Hold violation condition is same as in the case of synchronous counter

NPTEL

Kuruvilla Varghese

Now so that is basically the timing. So when we may be the controller design we can take in the next lecture, so or maybe we can look at the this, so the questionnaires when you implement control algorithm we basically what we have is an input output timing diagram ok.

(Refer Slide Time: 51:04)



Controller Design

57

- Control Algorithm
  - When one try to implement a control algorithm using an FSM one need to decide sequence of steps for various input combinations, and the output at each step. This can be a textual description followed with a waveform
  - Aim of the design is to come out with the truth tables of NSL and OL. This couldn't be done easily from textual description and/or waveform to truth tables
  - Hence, we use a graphical tool called state diagram (like flow chart) to visualize the sequence of states, their transition based on inputs, and various outputs produced at each state.

NPTEL

Kuruvilla Varghese

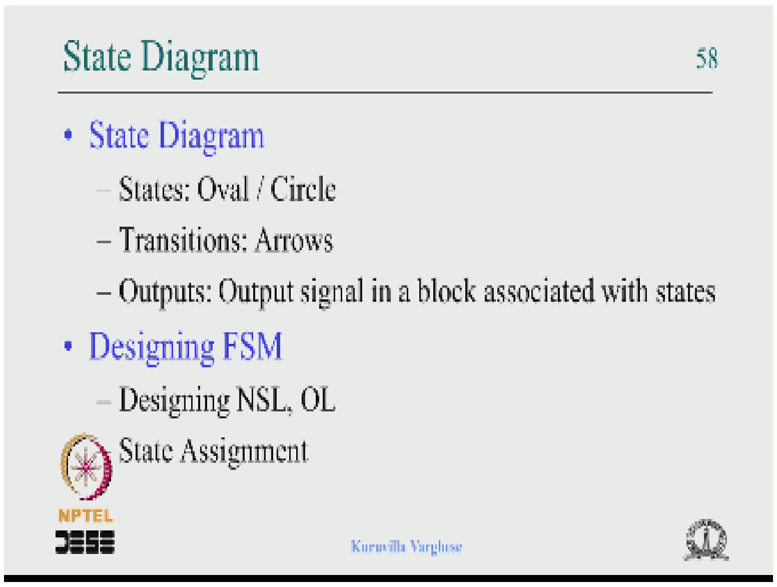
And from there we have to kind of decide a depending on the input what are the transition ok and at we needs transition what are the outputs required ok, normally I have a verbal description and a timing diagram from there we have to with respect to in both have to come out with various states sequences and output are each sequence and at that point that a possibility is not there only unique kind of way of solving a control problem.

So there could be different ways you can solve it something could be optimal. So essentially what I am saying is that it is very difficult to going one step from a waveform to an next stae logic and output logic. So we need to play with them and we need to design that to find algorithm and you know that in software before riding the code we have an algorithm then we used the flowchart and so.

When we try to think of control algorithm it is it is good if you can play with the graphical it is graphically. So that we can capture the basic idea. So similar to the flowchart we do something or less a diagram in designing this speed machine that is that helps us to kind of visualize the control algorithm to refine it, remove the redundancies and group them.

And all that you know that a thinking process with graphic symbols capturing basically the it should specify what are the inputs with respect input what are the transaction through the state and in each state what are the output this particular diagram should capture that we will see in the next class and it is called the state diagram or sometime it's called ASM chart algorithmic state machine whatever you call.

(Refer Slide Time: 53:18)



State Diagram

58

- State Diagram
  - States: Oval / Circle
  - Transitions: Arrows
  - Outputs: Output signal in a block associated with states
- Designing FSM
  - Designing NSL, OL
  - State Assignment

Kuruvilla Varghese

That is basically the idea of designing a control algorithm and that part we will handle it in the next lectures, so today what we have looked at basically is one quick run basically we have looked at an instruction call add A B and we have seen what are the essential macro steps involved in executing that and we have seen what is a corresponding waveform for that of the signals for that particular operation.

And we have laid out if possible waveform and we kind of walked out how this process can be generated and we said it cannot be generated out of a combination circuit we need a sequence and in this case we said that it can be generated out of a mod 3 counter and by decoding the present state and putting an output logic to decoded. The next question and that can be you know kind of implemented by writing the truth table.

And that is the truth tables seen and you can kind of a get a function output as a function of the present state then be the next question in last, and we have seen that this particular delay comes naturally because of the TCQ and the output logic and then we ask whether this is enough to generate all kinds of timing signals is a generic architecture. So we put some arbitrary waveform and we said yes it is fine because the that output logic can be changed the truth table can be changed to generate any kind of waveform in terms of the clock period.

Does not matter in this case the answer was yes and we have seen that answer we have seen that decoding and we also said that the more necessity that the state transition goes to a particular need sequence it can be any arbitrary sequence as long as we properly decoded. So there's no need to worry about how it comes as a counterpart but that can be made using minimising the area of logic and so on.

And essentially what we have is an input output waveform, so we come out with state transition and output ultimately from that we generate the next state table and output table and we have captured the finite state machine as a counter going to various case with an output logic to decode to generate output logic and we said that sometime it is convenient to decode output as a function of present state or input then it is called Mealy output.

If it is a decoder of present state is call Moore output and we also said there is another convenient way of looking at is combining these two blocks together because both get the same input. So that is shown here where the logic generate the next date and output and the last thing we have looked at is the timing issue which is similar to count but additional parties and output that is captured in the and the clock period.

Definitely it comes for I mean that has to be analysed because this output can go through another combinational circuit and register in that is this has to be modified, so which looks similar to the to the count of because after all the counter is a finite state machine okay. So counter is also a specific example of finite state machine output is not decode ok. So now do not like do for illustrating I said I can walk from the counter to the state machine ok do not think that the state machine is a counter.

But its other way in a state machine now state machine is more generic we can say asynchronous counter is a finite state machine is a specific case of a finite state machine not the other way just for grasping it. You know from kind of evolving it from the counter is convenient that is why I have discussed this way, but I don't say that the finite state machine is a counter.

A counter is a finite state machines are the way, so please now is very important topic once you grass this is basic architecture the timing you can master the design this is very minute I am very much lacking in textbooks because text books to show some practical cases it takes a lot of pages and pages. It is very difficult to kind of describing textbook that is why you could handle it, so please revise this, go through the reference book understand it well, we can continue in the next class with the designing. So I wish you all the best and thank you.